AFRL-IF-RS-TR-2001-163
Final Technical Report
August 2001

PARADISE AND DIRECT BROADCAST SATELLITE: A SOLUTION TO BATTLEFIELD DATA DISSEMINATION FOR THE 21ST CENTURY

University of Wisconsin

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F078

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

AFRL-IF-RS-TR-2001-163 has been reviewed and is approved for publication.

APPROVED: Craig S. Anken

CRAIG S. ANKEN
Project Engineer

FOR THE DIRECTOR: [illegible]

MICHAEL L. TALBERT, Technical Advisor
Information Technology Division
Information Directorate

PARADISE AND DIRECT BROADCAST SATELLITE: A SOLUTION TO BATTLEFIELD DATA DISSEMINATION FOR THE 21ST CENTURY

David Dewitt, Jeffrey Naughton, and San Dao

Contractor:	University of Wisconsin
Contract Number:	F30602-97-2-0247
Effective Date of Contract:	1 April 1997
Contract Expiration Date:	28 February 1999
Short Title of Work:	Paradise and Direct Broadcast Satellite: A Solution to Battlefield Data Dissemination for the 21st Century
Period of Work Covered:	Apr 97 - Feb 99

Principal Investigator:	David Dewitt
Phone:	(608) 262-5776
AFRL Project Engineer:	Craig S. Anken
Phone:	(315) 330-2074

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Craig S. Anken, AFRL/IFTD, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE AUGUST 2001	3. REPORT TYPE AND DATES COVERED Final Apr 97 - Feb 99

4. TITLE AND SUBTITLE PARADISE AND DIRECT BROADCAST SATELLITE: A SOLUTION TO BATTLEFIELD DATA DISSEMINATION FOR THE 21ST CENTURY	5. FUNDING NUMBERS C - F30602-97-2-0247 PE - 62301E PR - IIST TA - 00 WU - 11
6. AUTHOR(S) David Dewitt, Jeffrey Naughton, and San Dao	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Wisconsin Computer Science Department 1210 W. Dayton Street Madison WI 53706	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington VA 22203-1714; Air Force Research Laboratory/IFTD 525 Brooks Road Rome New York 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2001-163

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Craig S. Anken/IFTD/(315) 330-2074

12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This paper describes results from research under the Battlefield Aware Data Distribution (BADD) project. The goal of this research is to find new approaches to intelligent data distribution from distributed servers to roaming clients whose needs evolve with time. The primary focus of this report is on the part of the problem associated with the development of satellite-based information dissemination network infrastructure and smart push and filtering techniques. Research focused on possible ways to merge queries based on more abstract ideas, such as through generalization in a concept hierarchy. A number of issues in client cache management were also explored.

14. SUBJECT TERMS Distributed Data Dissemination, Networks, Multicast, Profiles	15. NUMBER OF PAGES 36
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Introduction

This report summarizes the final accomplishments of our Battlefield Awareness Data Dissemination (BADD) project.

1. Technical Issues and Accomplishments

The overall objective of HRL part of the BADD project is the development of satellite-based information dissemination network infrastructure and smart push and filtering techniques. Toward this end, in the final quarter, the majority of effort was in the following areas. First of all, HRL explored and demonstrated possible ways to merge queries based on more abstract ideas, such as through generalization in a concept hierarchy. In addition, HRL explored a number of issues in client cache management, both in efficient use of local storage as well the way it interacts with multicast dissemination of relevant data. Finally, HRL served as the site for the final presentations and demonstrations to the DARPA program management.

2. Semantic Merging in BADD

Earlier in this project, we implemented tools to merge two $\overset{\text{or}}{\wedge}$ more SQL queries based on straightforward structural modification of the queries and set-oriented approximations of a union of the results which are described by the component queries. For example, ranges of arithmetic values are combined and regions of a map are replaced by a polygonal region surrounding the components. While this provides an efficient framework for carefully structured data and queries, we were also asked to study possibilities for clustering requests and profiles based more on the conceptual aspects of the users' profiles. A *Profile* denotes a user interest. By *Merging of Profiles,* we mean combining several profiles into one profile. In this report we explore merging of profiles based on the semantics of the profile. Given a world view in the form of a class/type hierarchy, a sequence of user profiles each specifying a subset of types taken from that hierarchy, and a cost function defined on these profiles, the goal is to put these profiles in groups in a way that minimizes the total cost. This report discusses some of the cost functions and algorithms and the problems with them. We also point out the architectural issues in the integration of this scheme with *Paradise,* which was our backbone database engine in the implementation. Finally, caching of data is briefly touched upon.

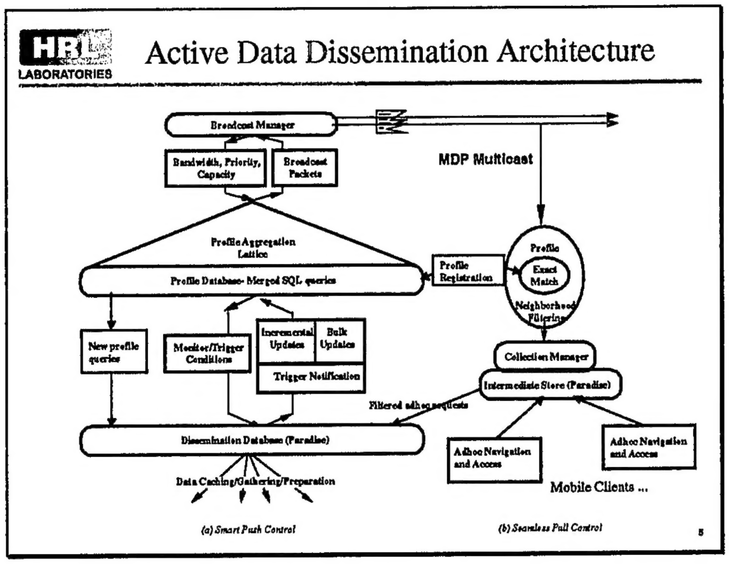

HRL LABORATORIES
Active Data Dissemination Architecture
Broadcast Manager
Bandwidth, Priority, Capacity
Broadcast Packets
MDP Multicast
Profile Aggregation Lattice
Profile Database- Merged SQL queries
Profile Registration
Profile
Exact Match
Neighborhood Filtering
New profile queries
Monitor/Trigger Conditions
Incremental Updates
Bulk Updates
Trigger Notification
Collection Manager
Intermediate Store (Paradise)
Filtered adhoc requests
Dissemination Database (Paradise)
Adhoc Navigation and Access
Adhoc Navigation and Access
Mobile Clients ...
Data Caching/Gathering/Preparation
(a) Smart Push Control
(b) Seamless Pull Control
5

Figure 2.1 Active Data Dissemination Architecture

2.1 Introduction

The *Battlefield Aware Data Distribution* (BADD) project at HRL in collaboration with University of Wisconsin-Madison aims at finding new approaches to intelligent data distribution from distributed servers to roaming clients whose needs evolve with time. Each client submits a **profile** to the server and the server sends the data that *matches* the profile to the client. The client has to submit the profile only once, while the server keeps sending data whenever relevant new data arrives. Also, the server sends only **new** data.

Since many clients may be interested in the same data or notification of the same events, the server multicasts the data. With present technology, the total number of multicast groups handled efficiently is limited – it is necessary to choose between the modest number supported by most networking hardware or software filtering of the full bandwidth. Hence in order to support thousands of clients, we need to merge profiles into only a few groups.

The challenges here are:

- A suitable language to describe the **profile**
- A way to understand the semantics of the profile
- Define a cost function on profiles
- Merge profiles efficiently

In the earlier phases of the project, the profile language was Paradise SQL. However, extracting higher level semantics from SQL is difficult, since it describes only the structure of the data, not its meaning.

An example of a cost function is the amount of extraneous data each client receives. This cost function can only be estimated since the relationship between a profile and the actual amount of data which matches can vary widely.

Each profile has an associated geographic region of interest, hence merging of profiles can be based in part on these regions, or the location of the clients. The regions will often show some useful correlation with network routing to clients and possibly with mission profiles.

2.2 Motivation for Semantic Merging

If a client is interested in movements of enemy tanks, and if there are enemy soldiers around, perhaps very soon the client will be interested in the movements of both the infantry and the tanks. If a pilot is looking for a place to land, the server could supply him a location of a military airbase, an aircraft carrier, a civilian airport, or even a flat field or highway depending on urgency, amount of fuel left etc. If you ask for a map of the area around you, the reply could be a detailed multi-layer map with information on roads, railway tracks, rivers, bridges, buildings, and elevation contours, or just the road-map if the bandwidth is limited. Thus extracting the rich semantics underlying the profile is an important task. Clearly this task is a formidable one if the profile is just an SQL query!

To describe the semantics of any object, we use *Ontology Markup Language* (OML)[1]. OML is a variant of XML in which one can describe objects, their conceptual inheritance and their relationships. There is no fixed DTD for an OML instance because in describing a new ontology framework, new classes of XML objects are introduced. We didn't find any OML parsers but after reading the OML definition document, we can still parse the OML instance document with most XML parsers by not using DTD validation. We have written a rudimentary parser suitable to our needs. The code is in /homes/akash/demo/server/oml/ParseOML.java. This java program uses XML parsers[2] provided by SUN's Java Project X, technical release 2, then adds some OML-specific postprocessing. Examples of OML definition documents can be found in /homes/akash/demo/ontologies/* and examples of OML instance documents in /homes/akash/demo/input/examples/*.

2.2.1 Profile Language

We restrict our language to one suitable for a battlefield. Every profile is an XML document with two main elements: "profile-region" and "profile-send". *Profile-region* describes the geographical region the user is interested in. This region has to be some type from the "region.oml" (an ontology for regions), e.g. circle or poly. *Profile-send* describes the types of objects in which the user is interested. They come from "defense.oml" (an ontology of defense, army, air-force, navy etc). This is just a set, e.g.

[1] see http://wave.eecs.wsu.edu/CKRMI/OML.html and http://ksl-web.stanford.edu/kst/ontology-sources.html
[2] see http://developer.java.sun.com/developer/producs/xml/ and file:/homes/akash/xml/xml-tr2/docs/api/index.html

```
<profile-send> <tank/> <plane/> </profile-send>
```

denotes that the user is interested in any data associated with any types of tanks and planes. Since **fighter-plane** is a subtype of **plane**,

```
<profile-send> <plane/> <fighter-plane/> </profile-send>
```

is equivalent to

```
<profile-send> <plane/> </profile-send>
```

In other words, each ontology is a tree structure in which each node includes all inferior nodes. When you select a node for a profile, you have selected all the nodes in the subtree below it.

The language described here is far from complete and many interesting questions cannot be expressed. But it was chosen for its simplicity and adequacy in demonstrating the concepts involved in semantic merging.

2.2.2 Cost function

To evaluate the effectiveness of any algorithm for merging profiles, a useful tool is to estimate some cost associated with merging and its effectiveness. E.g. a cost of bin packing is the number of bins used. Once we have defined a cost function, we compare the cost of the algorithm to the cost of the optimum solution (the minimum possible cost for the given input). Note the choice of various cost functions can lead to different optimal solutions, and may require different algorithms to attain optimal results.

Suppose a profile includes types a, b, c from the ontology tree. Let p be the nearest ancestor of all these. We define the *size* of the profile as $d(p,a) + d(p,b) + d(p,c)$, where distance d is the distance between two nodes along the tree edges. The distance between two profiles P,Q is then the size of the profile $P \cup Q$, ignoring all the spatial attributes.

Now the cost of merging can be defined as follows. Suppose the algorithm puts all the profiles into k multicast groups so that finally the *merged* profiles are $P1, P2,\ldots,Pk$. The cost of this merging is $max\{ size(Pi) \}$ where max is taken over $1 \leq i \leq k$.

2.2.3 Problem Definition

The *off-line* version of our problem is as follows: we are given all the profiles in the beginning. We have to put these profiles into k multicast groups minimizing the cost defined above.

While the off-line version is attractive because of its simplicity (and challenging because it is *NP-hard*), it does not really fit the BADD scenario. In BADD, clients submit or revise their profiles whenever they like, i.e. profiles come to the server at irregular intervals and the server has to immediately assign it to a multicast group without knowing any future requests from other clients. Clearly this lack of knowledge of the future imposes limitations on the effectiveness of any algorithm. This formulation of the problem is termed *on-line* and the measure of effectiveness of an *on-line algorithm A* is the *competitive ratio c(A)* defined as *max A(I)/OPT(I)* where *max* is taken over all inputs *I*, *A(I)* is the cost of algorithm *A* on *I* and *OPT(I)* is the cost of *OPT* (one who knows the entire sequence in advance) on *I*. If $c(A) < c(B)$ then we believe that *A* performs better than *B* in practice. For example, in bin packing we know that *FirstFit* outperforms *NextFit*. In the discussion so far, we have considered only **insertion** of new profiles, but in our case we have to consider deletions as well.

Hence our problem formulation is *on-line*. The algorithm does not know the entire sequence of requests/profiles before hand, and has to assign a multicast group to each request as it comes. If it assigns *j*th request to *q*th multicast group, then the group is replaced by the union of all the profiles in it (both spatial and semantic merging). If you think of each multicast group as a bin and the size of the merged-profile as the number of items in the bin or the load of the bin, then the goal is to assign profiles to a fixed number of bins, minimizing the maximum load. Generally, in an on-line algorithm, an implementation attempts to approximate the solution generated by an off-line algorithm while keeping most of the existing assignments intact. B-trees are a well known example where the tree is usually modified locally to maintain global balance. Generally, we want to avoid performing a complete reevaluation of the packing, both because this is a very expensive operation, and because changing existing assignments adds a significant network-management overhead.

2.2.4 Merging Algorithm

We use a greedy approach. When a profile request arrives, it is assigned to the bin whose load is minimum after merging, i.e. if *k* bins have profiles *P1, P2,..., Pk* in them (these are merged profiles and not the original requests), and new profile is Q, then choose the bin that minimizes *size of* $\{ P_i \cup Q \}$ over all $1 \leq i \leq k$.

2.2.5 Alternate Approaches

In the literature, people have mapped each request into a very high dimensional Euclidean space. Since Euclidean space is a metric space, we can use many of the available algorithms for mapping points into *k* clusters in a way that minimizes the maximum diameter of any cluster. This problem is NP-hard even when you know all the points (requests) in advance. Approximation algorithms for the *offline* version and competitive algorithms for the *online* version have been recently found [1,2].

This approach is not very effective with our model since it is difficult to map a profile to a point in R^d. If we write down every possible type in a row and use a 0,1-vector to represent what type is present in the profile and what is not, then the distance is biased by how large the subtree below any type is. Note that the distance defined by us in section 3.2 is not metric since it is does not satisfy the triangle inequality; for example, let p, q, r be three profiles with r being subtype of q. Then $d(q,r) = 0$, and $d(p,r) > d(p,q)$. Hence it is not true that $d(p,r) \leq d(p,q) + d(q,r)$, as required by the triangle inequality. We cannot apply any known clustering algorithms when the underlying space is not a metric space.

2.3 Integration with *Paradise*

Events in our system can be vague, e.g. radar might detect an enemy plane but not the type of the plane or what weapons it is carrying. In such case, if a client had expressed interests in watching any enemy planes, the server should send him a message. Note that the message here is incomplete. Now at the client end, we want to store all the inputs into a database, but where should this input go? If we had a table for each type of plane (and table columns being attributes of that type of plane), then since we do not know the type of the plane we are looking at, we cannot store this input in the database!

Our solution is to make a table for each node in the given ontology tree and place the column for an attribute at the node closest to the root, where it appears. Thus for all the types of the planes, if the attribute "height" is common, it appears at the node "plane" and not at the subtypes (by type inheritance "height" is attribute of all the nodes below "plane"). Then when we get information for a plane, we insert a tuple in the table "plane" and a tuple each in each "ancestor table". To keep track of which tuple in the parent refers to which tuple in the child table, each table has two default columns: *tid* and *pid*, where *pid* refers to the tuple id in the parent table that is related to this tuple in the child table. Starting at any tuple at any node and chasing all *pid*'s we can construct the original input (which might have *nulls* in them for the attributes whose values are not known yet!). Unfortunately, this generally requires using outer joins, which may not be very efficient in SQL. Some implementations of relational databases are beginning to provide better support for handing this kind of attribute inheritance.

The ontologies themselves can be stored in the database (since the "tree" defines a simple binary relation), although this has not been done. The client reads the OML files and constructs appropriate tables on startup. The code for this is in directory /homes/akash/demo/common/, files Database.java and Database.cc. It uses the libraries and API provided by *Paradise* (see /homes/akash/badd/paradise/installed/ include and lib).

2.4 Caching

The server is typically multicasting lots of data to the clients and although disk space is very cheap nowadays, we still have need to decide on throwing out some old or unneeded data eventually. Because these data items vary in size and we may also assign different costs to different data depending on either its importance or cost to recreate. Under these assumptions, traditional least recently used (LRU) algorithms are inapplicable, or at least the standard proofs of its optimality are not valid because the assumptions made for LRU are that all the data items are of the same size and cost. But there are other algorithms [3] that do not make these assumptions and are competitive.

All these algorithms are based on the cache hits. In our case, when the client listens to a particular multicast group, it often receives more than what it has asked for. Such data is never accessed but is still cached with the hope that the client might change its profile in the near future. Now when we run out of the space, what do we throw out from the cache? All the data that is extra and that is never accessed will be treated in the same way by above algorithms, but it should not be. One possible way to discriminate is to look at the semantic distance of the unused data to the profile and throw out the farthest one. Also, some of the data that is relevant to the current profile may be such that it is accessed only once and some of the data may be extremely volatile so that asking the server again is better than searching the cache, validating it and finding it stale. Another problem is with the cost model itself. Normally we want to maximize the cache-hit ratio. Thus at the time of the profile change if the new profile data is not in the cache, it should be counted as a "page-fault". This model is too simplistic and can be easily solved. This model does not capture the partial availability of data.

Another effort in caching is the "soft caching" approach. This works for data that is in multiple layers such as images. When out of space, throw out the high resolution part of the image but keep the low resolution version (consider e.g. the layered gifs/jpegs). Here just counting the hits will not do.

Caching is still under investigation and our present implementation does not address it.

2.5 Conclusions and Suggestions for Follow-on Work

We have integrated syntactic and spatial merging into *SmartPush* and *SmartPull*, and have changed the dissemination by TCP to reliable multicasting by *mdp*. On a different note, we have used Java programs to write servers and clients that use spatial merging and reliable multicast and also do semantic merging using OML. These programs also interact with *Paradise* and we have taken a novel approach to support incomplete information by simulating type hierarchies in SQL.

Possible future work would be to prove any bounds on the algorithm for merging we have suggested in section 3.4, or explore alternatives. There is also a need for improvement in the profile language. A lot of research is still needed on caching.

2.6 References

[1] Charikar et. al. "Incremental Clustering and Dynamic Information Retrieval", *Symposium on Theory of Computation, 1997.*

[2] Charikar et. al. "A constant factor approximation algorithm for the k-median problem", *Symposium on Theory of Computing 1999.*

[3] Young, N. "Online File Caching", *ACM-SIAM Symposium on Discrete Algorithms, 1998.*

3. Summary of Wisconsin Technical Results

Wisconsin's focus during the course of this project can be divided into two major parts. First, we designed and implemented a 3-tier implementation of the Paradise database system for use in a BADD environment. Second, we added a number of features to Paradise including support for triggers, new data types, and set-valued attributes. Finally, we served in a support role to HRL's prototyping efforts.

3.1 Three Tier Paradise Implementation

The main focus of the Wisconsin effort was the design and implementation of the 3-tier implementation of Paradise as shown in Figure 3.1 below. The bottom tier, labeled the *Paradise Fusion Server*, consists of a Paradise DBMS server process and two client processes (written using the Paradise cursor class library). The Smart Push process is responsible for packaging and disseminating information packages according to the profiles of the *Paradise Deployed Servers.* The Ingest Process is responsible for accepting updates from external data sources.

The middle tier consists of one or more *Paradise Deployed Servers (PDS).* Each PDS consists of a Paradise DBMS server process and a client process running the Smart Pull Software. The Smart Pull process is responsible for accepting disseminated information packages that it receives, filtering out data that it is not interested in. It is also responsible for aggregating profiles from multiple clients. The third tier consists of clients accessing their local Deployed Server using either one of the standard Paradise graphical user interfaces or a web-based browsing tool.

While we had hoped to test the prototype using a DirectPC GBS link, we were not able to arrange this. The prototype that we demonstrated uses sockets and TCP/IP to communicate. We also assume that there is a low-bandwidth connection between Deployed Servers and the Fusion Server that the Deployed Servers can use to update their profiles. The communication between Deployed Servers and their associated Clients is assumed to be a normal symmetric connection although it may be wireless in practice.

Operational Overview

The Fusion Server maintains one or more profiles for each of the deployed servers. The profiles are stored as stored procedures written in SQL with one or more parameters. As needed, Deployed Servers can update their profiles by sending new parameter values. Associated with each profile is a timestamp indicating the last time the profile was executed. Profiles stored at the Fusion Server will generally (but not necessarily) correspond to profiles that have been aggregated from the information needs of multiple clients by the corresponding Deployed Server. A similar table exists at each Deployed Server except that in this case the profiles probably have not been aggregated.

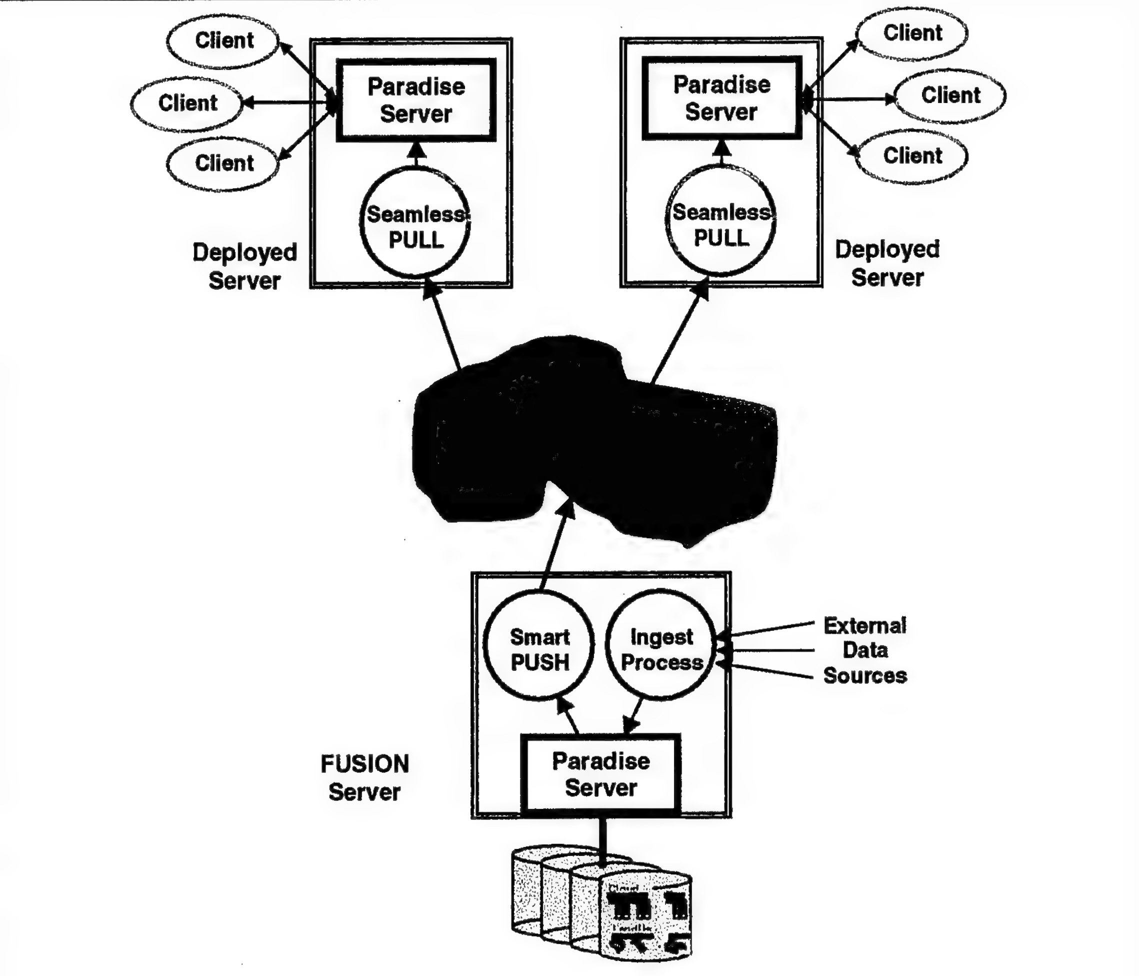


Figure 3.1

We implemented two mechanisms for controlling when data is disseminated. The first employs the trigger mechanism that we added to Paradise as part of this project. With this approach, as new tuples are inserted into the database at the Fusion Server, those profiles effected by the insertion are executed automatically. The second mechanism we implemented used timers to control the execution of each profile. With this approach each profile has a time interval associated it. This interval is used along with the timestamp indicating when the profile was last executed to determine when the profile should be executed next. Each approach is viable. For certain information, time is critical and the trigger-based approach will insure immediate delivery. For less-time critical information or in the case that satellite channel bandwidth is limited or needs to be carefully controlled, the timer-based approach to profile execution permits more careful control of the timing and utilization of bandwidth

While profiles are expressed as SQL queries, one does not want to disseminate the table resulting from the execution of a profile. Instead, what should be disseminated are the tuples from the base relations used in the query corresponding to the profile. Using the mechanisms described in the following section, the execution of a profile produces one or more tables of tuples to be disseminated. The Paradise unload utility is then used to dump each table to the file system. The result of this step consists of one file for the tuple data and one file for each large object (e.g. a SAR image). The Unix tar utility is then used to combine these files along with the schema information for the table to produce one large file. This file is then disseminated to the relevant Deployed Servers via the simulated GBS facility. As the file is received it is stored in the file system, untarred, and then the tuples are loaded into the database resident at the deployed server.

The advantage this approach is that the Paradise unload utility produces files that can be easily transferred to a Deployed Server and loaded. While it might be more efficient to pipeline the tuples resulting from the execution of a profile through the Smart Push process and onto the broadcast stream. However, doing this would require that the Smart Push software be capable of externalizing all types of large objects (large raster images, video, very large polygons, etc.

Computing What to Disseminate

One of the key technical challenges of this environment is determining what data needs to be disseminated. The obvious approach of sending data that has been inserted since the last time the profile was executed turns out to not be correct. In particular, the insertion of new data can cause "old", undisseminated data to become relevant to a Deployed Server. In the remainder of this section we describe the approach that we have developed for determining what data needs to be disseminated.

Consider the following two tables on Landings and Takeoffs. In addition to information on planes, each table contains timestamp (TS) and unique identifier (ID) fields. The timestamp field is used to indicate the time at which the tuple was inserted into the table (Please note that the following data was contrived to make the examples work out. In general, the ID values must be unique)

Landings

TS	ID	Plane
15	2	b21
25	3	b22
15	4	b23
25	5	b24

Takeoffs

TS	ID	Plane
10	2	mig21
15	3	mig22
30	4	mig23
35	5	mig24
17	4	mig25

Consider the profile:

SELECT * from takeoffs, landings where landings.ID = takeoffs.ID

(This profile makes little or no sense but it serves to illustrate the process of determining what new information needs to be disseminated).

When stored at the Fusion Server this profile gets rewritten as a series of queries (X represents the time the profile was last executed).. The first query, Q1, is shown below:

```
Q1: SELECT
        landings.ID AS LandingsID,  landings.TS AS LandingsTS,
        takeoffs.ID AS TakeoffsID, takeoffs.TS AS TakeoffsTS,
        (landings.TS<=X AND takeoffs.TS<=X) AS Old

    INTO R1
    FROM landings, takeoffs WHERE landings.ID = takeoffs.ID
```

For X=20, Q1 produces the following result relation R1:

LandingsID	LandingsTS	TakeoffsID	TakeoffsTS	Old
2	15	2	10	1
3	25	3	15	0
4	15	4	30	0
5	25	5	35	0
4	15	4	17	1

The meaning of Old in this table is as follows. Old == 1 if both the corresponding Takeoffs and Landings tuples were inserted into the database **prior** to the last execution (i.e. time 20) of the profile. In this case, neither tuple needs to be disseminated. Old == 0 if at least one of the two joining tuples was inserted after the previous execution of the profile (ie. time 20). In this case, both tuples need to be disseminated even if one of them happens to be "old." This is the key to our approach for automatically determining what data needs to be disseminated. This approach avoids disseminating tuples that are not of interest to the Deployed Server while insuring that all relevant tuples are properly disseminated in a timely fashion.

The next step in the process is to execute the following query (This process actually gets repeated for each base table in the original profile):

Q2: SELECT R1.LandingsID, R1.Old INTO R2 FROM R1;

Q2 produces the table R2 as shown below:

LandingsID	Old
2	1
3	0
4	0
5	0
4	1

The rows of this table are interpreted as follows. Tuple 3 (ie. LandingsID == 3) and Tuple 5 should both be disseminated. Tuple 3 (which is new) joins with an old tuple from the Takeoffs table while Tuple 5 joins with a new tuple from the Takeoffs table. Tuple 2 (which is old) joins with an old tuple from Takeoffs and should not be disseminated.

The interesting thing to note is that the LandingsID value 4 appears with Old values of both 0 and 1. Note that this Landings tuple with ID=4 (see Landings table above) was inserted into the database at timestamp 15 – prior to the previous execution of the profile. The first case (i.e. Old == 0) means that the tuple joined with a new (i.e. TS > 20) Takeoffs tuple and is a candidate for dissemination. The second case (i.e. Old = 1) means that the tuple joined with an old (i.e. TS <= 20) Takeoffs tuple and hence has already been disseminated. The next step is to run the following query which will automatically distinguish between these two cases so that we can avoid disseminating the tuple a second time).

Q2Prime:

```
SELECT R2.LandingsID, Max(R2.Old) AS MaxOfOld into R2Prime
FROM R2
GROUP BY R2.LandingsID;
```

This query produces:

R2Prime

LandingsID	MaxOfOld
2	1
3	0
4	1
5	0

Now we can clearly see that tuples 3 and 5 should be disseminated while 2 and 4 should not be.

As an optimization to this two step process, Q2 and Q2Prime can actually be combined into a single SQL query:

```
Q2ALL:  SELECT R1.LandingsID, Max(R1.Old) AS MaxOfOld
        INTO R2Prime
        FROM R1
        GROUP BY R1.LandingsID;
```

The final step in the process is to use the following query to select the proper tuples from the Landings table to disseminate:

```
Q3:
    SELECT Landings.*
    FROM Landings, R2Prime
    WHERE Landings.ID = R2Prime.LandingsID
    and R2Prime.MaxOfOld = 0;
```

This query produces the following result table of tuples from the Landings relation that need to be disseminated

TS	ID	Plane
25	3	b22
25	5	b24

This process (Q2, Q2Prime, and Q3) is then repeated for each of the other base tables in the original profile Q1. In this example, the only other base table is TakeOffs. The equivalent sequence of queries yields the following tuples from the Takeoffs relation that need to be disseminated. Please observe that the first tuple was inserted into the database prior to the previous execution of the query and that it is being disseminated at the current time because it joins with a newly inserted tuple from the Landings table.

TS	ID	Plane
15	3	mig22
35	5	mig24

This mechanism has been implemented in our three-tier Paradise implementation. Profiles submitted by a Deployed Server are automatically translated into the appropriate sequence of SQL queries. This sequence of queries is stored, along with the profile definition, in the database.

Using XML as a Dissemination Mechanism

We also explored the use of XML as a vehicle for disseminating information between the Fusion Server and the Deployed Servers. In the current prototype, after determining which base-relation tuples need to be disseminated, a temporary table containing the tuples is created. The Paradise unload utility is then used to dump the table to the file system. The result of step consists of one file for the tuple data and one file for each large object (e.g. a SAR image). The Unix tar utility is then used to combine these files along with the schema information for the table to produce one large file. This file is then disseminated to the relevant Deployed Servers via the simulated GBS facility. As the file is received it is stored in the file system, untarred, and then the tuples are loaded into the local database at the deployed server.

While this approach works correctly, its batch-oriented approach limits its efficiency. In addition, the load files produced for distribution are in a Paradise-specific format. Thus, if a Deployed Server is using a database system other than Paradise, conversion software must be written to convert the disseminated files to the appropriate format.

As an alternative, we explored the use of XML as a protocol for communicating between the Fusion Server and the Deployed Servers. In addition, since web-browsers are becoming XML enabled, XML could also be used by clients wishing to use a web browser to browse the database stored at their local Deployed Server.

We also designed and implemented software running as a client-application to Paradise that will converts the tuples returned as the response to a query into an XML-compliant form. In addition, we defined DTDs (data type definitions) for all Paradise's extended types including spatial, images, arrays and video as well as the standard base types.

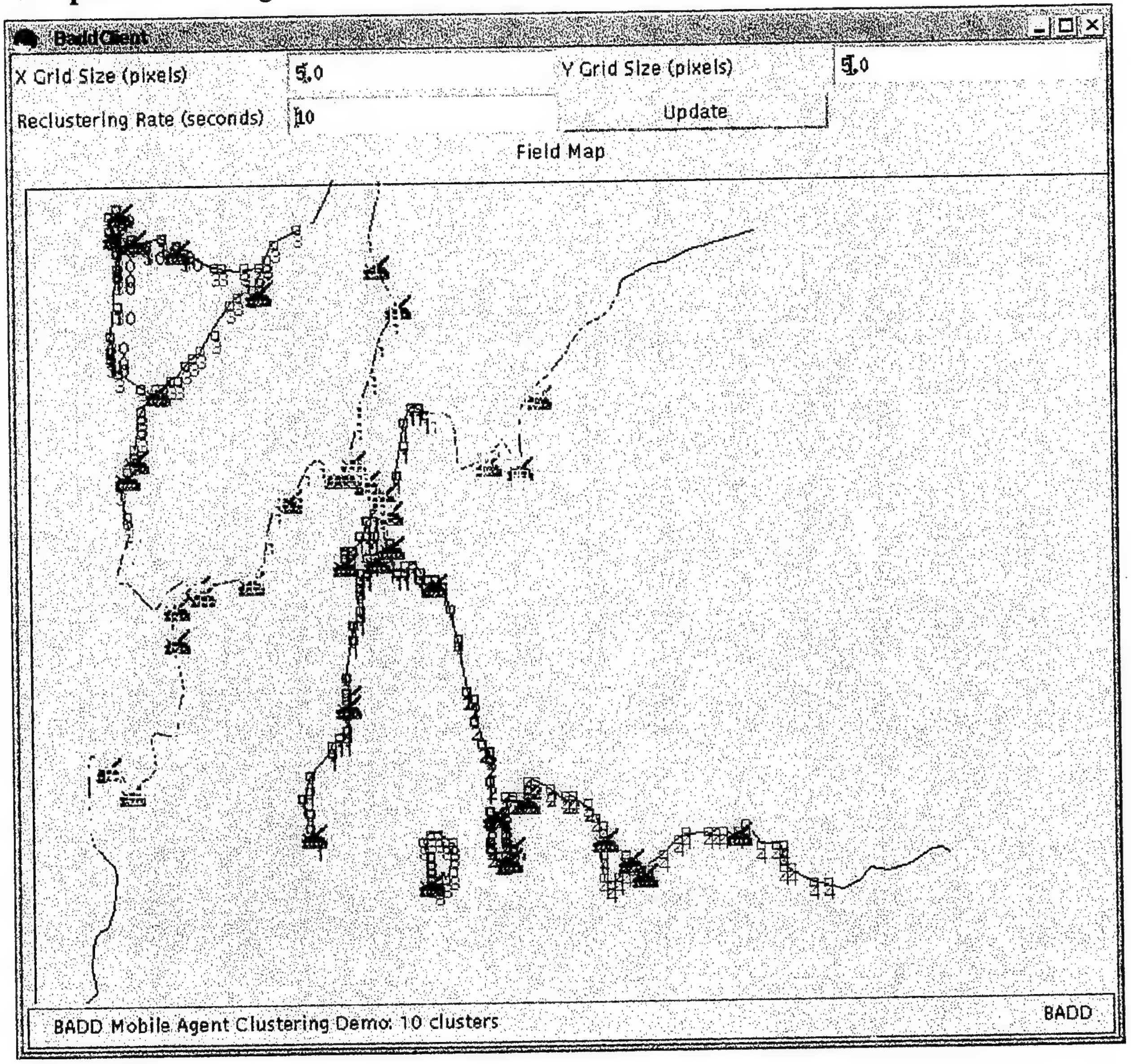
3.2 Technical Enhancements to Paradise

As part of this effort we also added a number of new features to Paradise including support for triggers, set-valued attributes, and text and temporal data types. Each of these features was described in an earlier progress report and/or technical report that have been submitted for publication.

4. Final prototype demonstrations at HRL

We concluded this project with a technical presentation at HRL on September 27, 1999 to Ray Liuzzi from the Air Force Research Laboratory in Rome, N.Y. and Colonel Riki Barbour from DARPA. In addition to describing our results, we also gave of number of demonstrations of the technology developed as part of this project.

4.1 Spatial Clustering Simulation



X Grid Size (pixels) 5.0
Y Grid Size (pixels) 5.0
Reclustering Rate (seconds) 10

BADD Mobile Agent Clustering Demo: 10 clusters

In this demonstration, we showed the graphic display component of the simulation of road-based clustering of a collection of mobile agents. This simulation serves to show some of the issues and tradeoffs in trying to effectively cluster a collection of mobile data clients when their movements are often constrained by terrain such as mountains and rivers, and by man-made features like roads and bridges. The work behind this and the results of the simulations were extensively discussed in the January 1999 quarterly report.

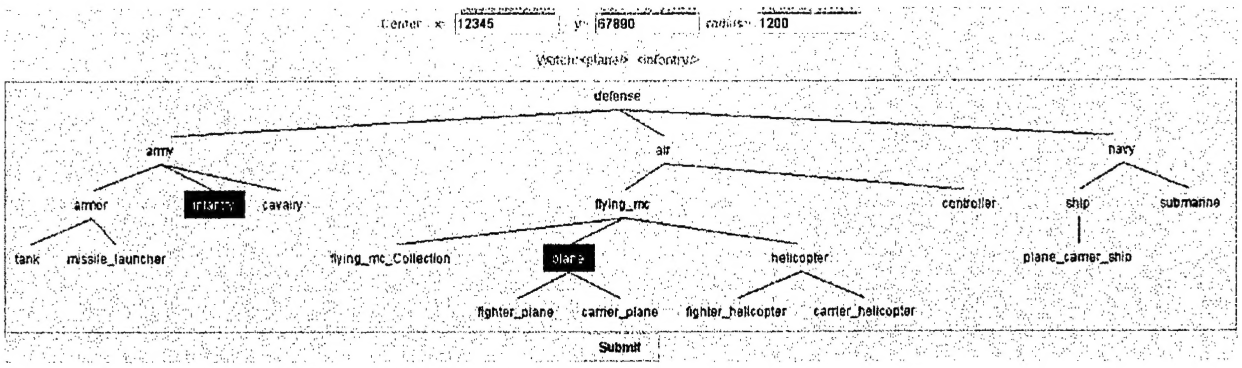
4.2 Paradise database integration

Based on the demo scenario from U. Wisconsin, this demonstrated used Paradise graphical user interface to display events as they happened, one screen for each site. The registration of interest profiles was routed through a new profile manager which was setup to merge overlapping circles of interest into a polygon approximating both, as well as combining other query features so as to present a single merged profile to Paradise. Details about the query merging algorithms were mainly reported in the April and July 1999 quarterly reports. In addition, the data dissemination from the server was done using the mdp reliable multicast code. Mdp is a Multicast Dissemination Protocol implemented by the Naval Research Laboratory. In the prototype, we were using Version 1.0 Alpha. MDP is one example of a number of reliable multicast implementations under current development in the internet community. MDP is oriented toward distributing a collection of discreet files. Our use of MDP was primarily motivated by its availability in source form, simplifying customization, and by the few restrictions on its use. More information about MDP can be obtained from the developers/maintainers at NRL, Joe Macker macker@itd.nrl.navy.mil and Brian Adamson adamson@itd.nrl.navy.mil.

4.3 Conceptual clustering

In this prototype simulation, HRL demonstrated the concepts discussed above in Section 2.2. When each client starts, the first step is to read the (OML) ontology describing its universe. It then creates a database in Paradise with one table for each node in the ontology. Finally it displays a graph of its ontology in one window for user control, and opens a second window for displaying distributed data received on its multicast channel. When a user selects nodes for his profile and clicks the Submit button, the profile is sent to the server process. Figure 4.3.1 shows a client who has highlighted infantry and plane. As the server multicasts OML clauses to the clients, the client uses the information in the message to choose an appropriate icon and the coordinates at which to display it. Figure 4.3.2 shows the two active targets matching the profile and falling within the stated circle. The display for a second client is not shown, but at the time of the snapshot, none of the targets were both inside the coverage area requested and matched its profile.

The server process uses the client profile registrations to construct a set of multicast groups, and as data events are received, the data is compared to each group profile, and when matched, sends a multicast data block to that group of recipients. Figure 4.3.3 shows the server map display with the polygonal approximations of the coverage area of the two clients active at the time of the snapshot, plus the four targets currently active. At this moment, only the infantry and plane data is being multicast as relevant to any user. The second client is suppressing the display of these items because the do not fit its profile, but both clients are storing the multicast reports in their local databases.


Center x: 12345 y: 67890 radius: 1200

Watch<plane> <infantry>

Figure 4.3.1 Client profile configuration interface

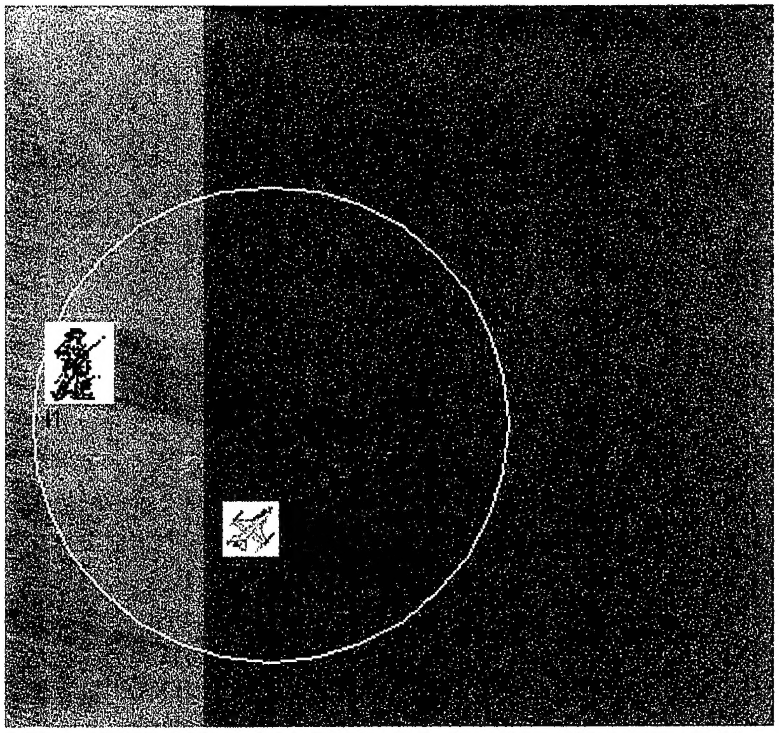


Figure 4.3.2 Client "map" display showing objects matching profile and area of interest.

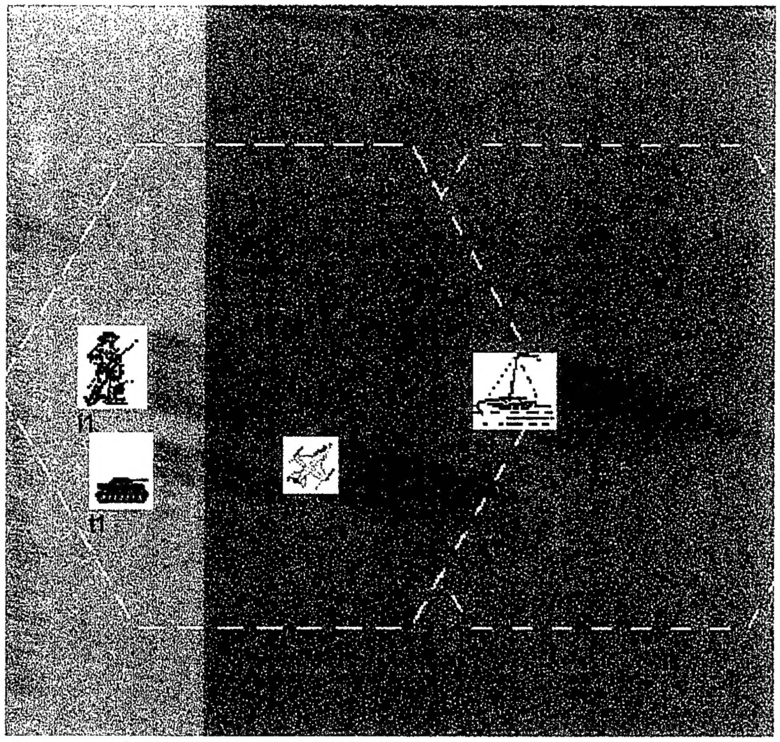


Figure 4.3.3. Server "map" display showing areas for two clients and current reports.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.